A TRIDENT SCHOLAR
PROJECT REPORT

NO. 251

"Speech Coding and Phoneme Classification Using a Back-propagation Neural Network"

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

20031201 113

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including g the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 7 May 1997	3. REPORT TYPE AND DATE COVERED

4. TITLE AND SUBTITLE
Speech coding and phoneme classification using a back-propagation neural network

5. FUNDING NUMBERS

6. AUTHOR(S)
Brett A. St. George

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Naval Academy
Annapolis, MD

8. PERFORMING ORGANIZATION REPORT NUMBER

USNA Trident Scholar project report no. 251 (1997)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Accepted by the U.S. Trident Scholar Committee

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT: Speech is a natural, unspecialized method of communication that is perhaps the ultimate machine interface. Previous attempts to provide such an interface, however, have been limited to pre-defined vocabularies of an artificial syntax. This paper presents a method for speaker-dependent speech identification that uses a back-propagation neural network to determine the phonemes present within a voice signal. The vocal tract changes slowly in time and can be modeled using the pitch and format frequencies of the voice. These frequencies relate the position of the vocal tract to specific pronunciations and are obtained by using a homomorphic filtering process that separates the vocal tract's impulse response from the excitation source. The frequency representation of this response is concatenated with the excitation containing the pitch frequency and applied to the input layer of the neural network. From this information, the network selects combinations of features that identify the phonemes which are present. This network was trained on a set of speaker dependent phonemes, and now phonetically classifies new speech input. This classification scheme could be used to translate linguistic messages into machine code with a very high data rate. This benefit would allow for real-time interaction with machines with no specialized training. Applications could be as simple as providing quick voice to text processing or as diverse as implementing a control system with response time tied to specified voice patterns.

14. SUBJECT TERMS
speech coding, cepstral analysis, homomorphic deconvolution, back-propagation neural network, phoneme

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500
(Rev.2-89)

Standard Form 298
Prescribed by ANSI Std. Z39-18
298-102

"Speech Coding and Phoneme Classification Using a Back-propagation Neural Network"

by

Midshipman Brett A. St. George, Class of 1997
United States Naval Academy
Annapolis, Maryland

_____
(signature)

Certification of Advisor Approval

Instructor Ellen C. Wooten
Electrical Engineering Department

_____
(signature)

_____
5/7/97
(date)

Assistant Professor Louiza Sellami
Electrical Engineering Department

_____
(signature)

_____
5/7/97
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Chair, Trident Scholar Committee

_____
(signature)

_____
7 May 97
(date)

Abstract

Speech is a natural, unspecialized method of communication that is perhaps the ultimate machine interface. Previous attempts to provide such an interface, however, have been limited to pre-defined vocabularies of an artificial syntax. This paper presents a method for speaker-dependent speech identification that uses a back-propagation neural network to determine the phonemes present within a voice signal.

The vocal tract changes slowly in time and can be modeled using the pitch and formant frequencies of the voice. These frequencies relate the position of the vocal tract to specific pronunciations and are obtained by using a homomorphic filtering process that separates the vocal tract's impulse response from the excitation source. The frequency representation of this response is concatenated with the excitation containing the pitch frequency and applied to the input layer of the neural network. From this information, the network selects combinations of features that identify the phonemes which are present. This network was trained on a set of speaker dependent phonemes, and now phonetically classifies new speech input.

This classification scheme could be used to translate linguistic messages into machine code with a very high data rate. This benefit would allow for real-time interaction with machines with no specialized training. Applications could be as simple as providing quick voice to text processing or as diverse as implementing a control system with response time tied to specified voice patterns.

Keywords: speech coding, cepstral analysis, homomorphic deconvolution, back-propagation neural network, phoneme.

Acknowledgments

I would like to thank God for giving me the patience, diligence, guidance, and reasoning to accomplish all that I have. I also offer my thanks to everyone whose support and encouragement have been my crutch throughout this project. Mom, Dad, and Holly, thanks for always being my sounding board. You didn't always understand my problems, but you never failed to see me through them. Amanda, your contribution is a book unto itself. Thanks for always being there. LT Reppar, if it were not for your support, I would have never had the opportunity to be a part of this program. Daphi, your countless hours spent fixing my computer are not forgotten. And lastly, my deepest gratitude goes to Dr. Antal A. Sarkady and to my advisors Dr. Ellen C. Wooten and Dr. Louiza Sellami.

Contents

Figures

Chapter I: Introduction

Speech coding and neural network processing are powerful tools that are empowering us to teach our computers to talk as well as teach them to listen to us speak. These tools have created a new horizon of opportunity that is allowing researchers to apply voice processing to many new areas of study. Some possible applications include conversing with computers over the telephone without the need of a modem, carrying a hand-held translator while touring a foreign country, or preparing a paper without the need of a keyboard. Speech-integrated circuitry has a wide variety of uses that are being incorporated into simple devices such as calculators, pocket dictionaries, and toys. Of the many benefits that voice processing provides, one of the most distinct advantages is that it allows for real-time interaction with machines with no specialized training. This advantage is creating a new method of computer interface that is prompting intense interest in the area of voice processing.

This paper continues to build on current voice processing research by developing a phonetic classification scheme that relies on speech coding and back-propagation neural network processing. In the first part of this chapter, speech coding is defined and an explanation is provided to describe why neural network processing provides a method which is very well suited for speech classification. Following this discussion, a description of the project is flowcharted and the contributions from this investigation are outlined.

1. Speech Coding and Neural Networks

Speech coding is an area of signal processing that involves extracting information from an analog, time-based signal, and improving the efficiency of speech transmission,

classification or recognition [1, p. 143]. The coding process incorporates several techniques for extracting information from a speech signal that is not apparent in the analog time domain representation of the unprocessed signal. This information reveals certain acoustic variables that describe the physical nature of the vocal tract for the pronunciation of various sounds. This precept has led to an intense study of phonetics using signal processing and has created a strong mathematical foundation for providing voice activated systems. Unfortunately, such systems have been confined to a specific user and have been designed to respond only to pre-defined vocabularies that are often artificial in syntax. Because of these limiting factors, neural networks have been introduced as a voice processing tool that will hopefully allow speech recognition systems to respond to more general inputs. Neural networks have the advantage of being adept at pattern recognition tasks and are often able to outperform both traditional statistical and expert systems [2, p. 8]. For this reason, a neural network recognition system was chosen for the phonetic classification scheme.

Of the many different neural network algorithms that provide some method of classification, a back-propagation neural network was specifically chosen because of its ability to match large amounts of input information simultaneously, and then generate categorical or generalized output [2, p. 8]. In the classification scheme presented in this paper, the output of the network corresponds to the phonetic alphabet as defined by 40 phonemes that were selectively chosen to represent the speech of a specific person. These 40 distinctive sounds can be combined to produce virtually all spoken words in the American-English language.

2. Project Description

The flowchart depicted in Fig. 1 describes the speech identification process. This procedure uses two software packages, MATLAB and NeuralWorks, to perform both the speech coding process and phonetic mapping. These processes, however, rely first on obtaining a sampled speech signal. This set of voice data is generated by speaking into a microphone connected to the computer. The microphone converts the speech into a voltage that is sampled at discrete intervals with a 16-bit analog-to-digital (A/D) converter that is intrinsic to the sound card within the computer. Once sampled, the speech signal is stored as Windows 'wav' file and is accessed in MATLAB as a data vector. A speech coding process is applied to this data vector and the resulting vocal tract function and pitch frequency are placed into the input layer of the neural network. The network is constructed using NeuralWorks and is able to phonetically classify segments of speech.
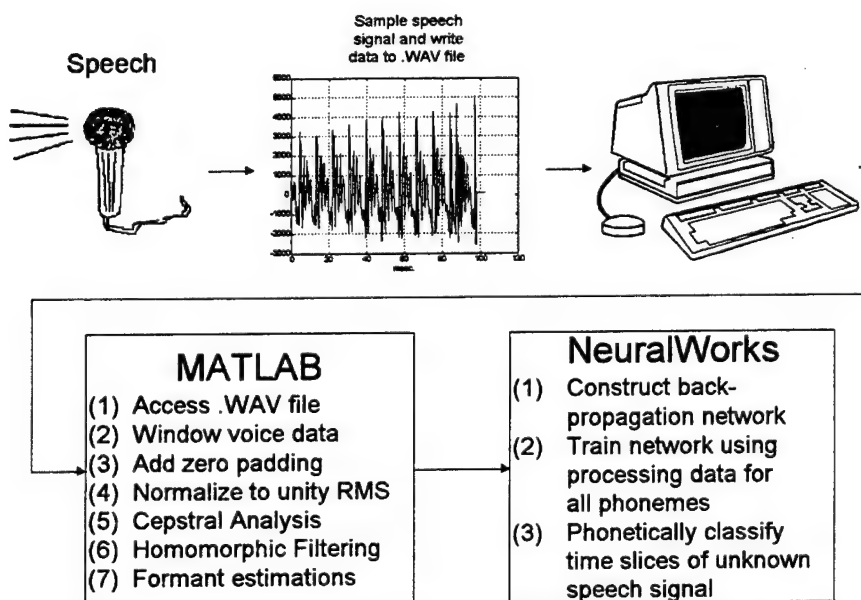

Fig. 1 Flowchart of speech coding and classification algorithm

3. Contributions

The back-propagation neural network constructed in this project incorporates many new design aspects that are new to the area of voice processing. A list of these specific contributions include:

- Determined 40 phonemes to describe the voice of the speaker

  Since the speech-identification algorithm developed in this paper is speaker dependent, it was first necessary to adopt a list of phonemes that could phonetically describe most spoken words. Several texts contain lists that describe the phonemes which comprise most of the American-English language; however, within each of these lists there is some disparity when regional dialects and accents are considered. To account for some of these possible ambiguities or redundancies that might arise, each phoneme was tested to insure that it could be defined by a distinct vocal tract shape. Ultimately, 40 phonemes were found to represent the voice of the speaker. Of these 40, one phoneme was defined as a null or a pause in speech when nothing is said.

- Constructed six data banks for training and classification

  Six data banks were constructed that each contained examples of the 40 phonemes. Three of these banks were used to train the network while the remaining three were reserved for testing its performance.

- Constructed vocal tract model to serve as input to the neural network

  Neural networks have been used in speech identification in the past, but these networks have been limited to classifying only vowels. Of these networks, the impulse response of the vocal tract has been used as the sole determinant in

distinguishing between different phonetic pronunciations. The input to the network developed in this project, however, includes the impulse response of the vocal tract and an estimation of pitch frequency describing the resonation of the vocal chords.

Also, there are a large number of variables that govern the speech coding process, each of which significantly affects the ability of the neural network to correctly classify speech. Some of these variables include record length, size and type of data window, and the number of ensemble averages. Consequently, appropriate values for these variables were determined through experimentation to optimize the performance of the network.

- Designed a back-propagation neural network to classify phonemes

Several parameters defined during the construction of the network such as the number of hidden layers, number of processing elements, learning coefficient, network range and momentum term were each individually tested. These variables have a distinct impact on the ability of the network to correctly classify the phonemes present within the speech signal. These values, however, must be assigned through a trial and error process based on classification results. Hence, several experiments were performed to determine the values assigned to these variables.

- Designed filtering routines to limit phonetic misclassifications

Within any sampled signal, there is inherently background noise present. To this noise, further ambiguity is introduced when slight variations occur within the vocal tract shape that can cause certain phonemes to have overlapping characteristics. This variation coupled with additive noise can mask the pertinent features of a speech signal and cause the phonemes within the signal to be misidentified. Filtering routines based on ensemble averaging were developed as a method for "smoothing

11

out" these ambiguities. Additionally, a statistical analysis was developed to remove both anomalies and redundancies in the output.

4. Overview

Based on the variety of applications for voice recognition and processing systems, this paper examines an algorithmic process for classifying phonemes within a speech signal. This algorithm employs a speech coding technique for finding the transfer function which effectively describes the physical attributes of the vocal tract. This information is then applied to a neural network to determine the phonemes present in the original voice data. In the next chapter, both the speech coding technique and back-propagation neural network algorithms are further discussed. In particular, Chapter II introduces homomorphic deconvolution and cepstral analysis as digital signal processing techniques for extracting the impulse response of the vocal tract from a time-based analog speech signal. These two techniques provide the information that is passed to the neural network and used in the phonetic classification scheme. In Chapter III, a simulation is presented showing the speech coding and classification process for an example phoneme. This chapter concludes with results from the application of these algorithms as applied to all 40 phonemes. Finally, Chapter IV completes the discussion by citing some general conclusions and areas for further work.

Chapter II: Speech Coding and Neural Processing

The speech identification process can be divided into two areas of study, speech coding and neural network processing. An introduction to each of these two areas, with supporting examples, is now provided.

1. Speech Coding

To understand the speech coding process, it is necessary to begin with a description of the physical nature of speech. Sound is produced when air is forced from the lungs and becomes filtered by variations in the vocal tract shape to produce a speech signal [3, p. 53]. These variations in shape determine the characteristics of the filtering function that shape the frequency spectrum of the final speech signal. If this filtering function can be directly extracted from a sampled speech signal, it can be used to identify which phonetic character is being pronounced. The rest of this section introduces a speech coding technique for extracting this filtering function.

To begin the speech coding process, suppose that the vocal tract acts as a linear time-invariant system within a sufficiently short time slice. This is a valid assumption for a short segment of speech where the vocal tract shape is unchanging and remains fixed in shape. Hence, the frequency representation of a speech segment, $F(j\omega)$, can be represented as the product of an excitation source, $E(j\omega)$, and a transfer function, $H(j\omega)$:

$$F(j\omega) = E(j\omega) \cdot H(j\omega) \tag{2.1.1}$$

To separate these two functions, the complex logarithm of Eq. 2.1.1 is used to create a real and imaginary parts that reflect the magnitude and phase angle respectively.

$$Ln(F(j\omega)) = Ln|F(j\omega)| + j\theta_{F(j\omega)} \qquad (2.1.2)$$

Because phase angle carries only information about the time origin of the signal, the imaginary part of Eq. 2.1.2 can be ignored [4, p. 360]. The real values, however, represent a magnitude function where the excitation source and transfer function are additive in the frequency domain [5, p. 266].

$$Ln|F(j\omega)| = Ln|E(j\omega)| + Ln|H(j\omega)| \qquad (2.1.3)$$

Taking the Inverse Fast Fourier Transform (IFFT) of the real portion of the complex logarithm, Eq. 2.1.3, produces a time domain signal in which the logarithm of the excitation source and impulse response are separable. This filtering process is known as homomorphic deconvolution and involves a technique referred to as "cepstral deconvolution." To employ this technique, the combined signals need to have their main components of energy concentrated at different frequencies [5, p. 266]. This condition is true of speech when only the transfer function and excitation source are considered. The transfer function describing the vocal tract, $H(j\omega)$, has a band-limiting characteristic that confines the energy of the speech signal within a 5 kHz range. Conversely, the excitation source, $E(j\omega)$, can be modeled as a white noise source which contains an equal distribution of energy across a frequency range far in excess of 5 kHz. Because the primary energy components of both functions do not overlap, cepstral analysis applies.

To implement this process, a second Fourier analysis is applied to the logarithm of the frequency spectrum. The resultant function is called the "cepstrum" (Eq. 2.1.4). In addition to separating the vocal tract response from the excitation source, the cepstrum provides a technique for classifying speech into two categories, voiced and unvoiced. During voiced speech the vocal chords vibrate at a constant frequency known as the fundamental or pitch frequency. Alternatively, in unvoiced speech the vocal chords do not move and air is forced past the glottis, tongue, teeth and lips. Both of the categories have distinguishable features that become apparent in the cepstrum. For voiced speech, the cepstrum contains a spike at integer multiples of the pitch period. This is not the case for unvoiced speech, since the vocal chords do not resonate and the cepstrum contains no noticeable spikes.

$$\text{cepstrum} = \text{IFFT}(\text{Ln}|F(j\omega)|) \tag{2.1.4}$$

Applying a rectangular time window to the real portion of the cepstrum separates the impulse function from the excitation source. The frequency domain representation of this filtered function reveals the formant frequencies. Formants are those frequencies that resonate within the vocal tract. These specific frequencies appear as resonant peaks in the frequency domain and provide an acoustic variable for relating the position of the vocal tract to a phonetic pronunciation.

During the windowing process described in the preceding paragraph several experiments were necessary to determine the size of the time window that should be applied to the real portion of the cepstrum. The size of this data window was decided

upon by making comparisons with various N-point moving average filters. Because cepstral deconvolution removes the high frequency components of a signal, a moving average filter produces very similar results and provides a measure for judging the effectiveness of the cepstral smoothing process.

In many voice processing applications, a moving average filter is used as an alternative to cepstral smoothing simply because it provides a faster processing rate. This advantage, however, is offset by the necessity to vary the width of the filter for signals having substantially different frequency spectrum. Cepstral deconvolution, on the other hand, naturally accommodates this width change. Fig. 2 shows two examples using both cepstral analysis and a moving average filter. In the examples the formant frequencies are shown for two separate phonemes. The phoneme "e" as in "bet" is a vowel produced by exciting a fixed vocal tract with quasi-periodic pulses of air caused by vibration of the vocal chords [4, p. 43]. Because of this vibration, certain frequencies resonate within the vocal tract and the frequency spectrum is naturally smooth. The phoneme "t", however, is an unvoiced stop consonant and is produced by a "brief interval of friction followed by a period of aspiration" [4, p. 53]. This property creates a frequency spectrum that shows substantial variations in spectral content. As a result, a moving average filter of varying width is required. As seen in Fig. 2, however, cepstral deconvolution eliminates this necessity.

Now that the speech coding process has been introduced, the discussion will focus on neural network processing. The next section will begin with a definition of neural
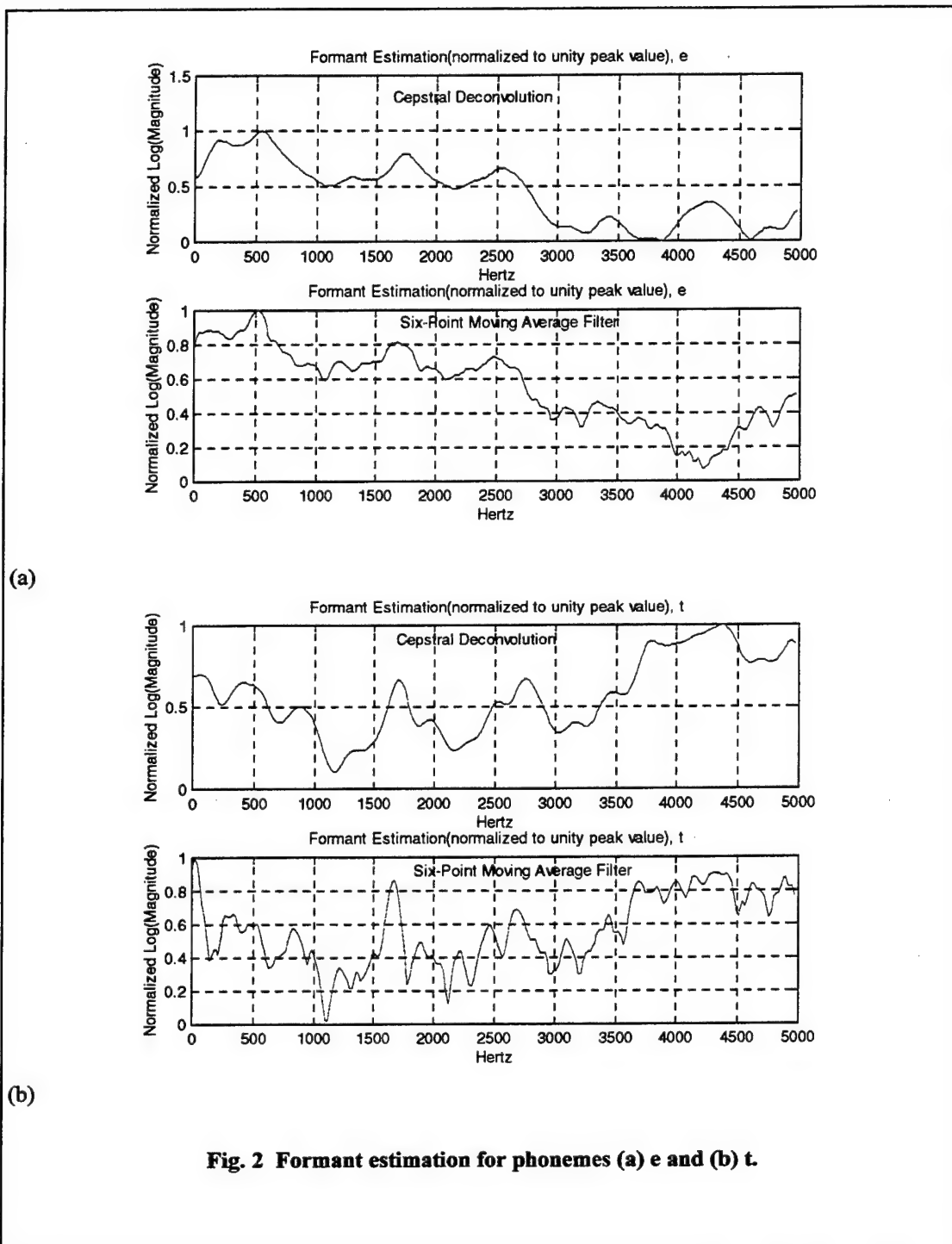
Fig. 2 Formant estimation for phonemes (a) e and (b) t.

networks as well as a description of their basic architecture. This explanation will be followed by a detailed discussion of the training algorithm that is used to actually "teach" the network to identify phonemes.

2. Neural Network Computing

(a) Introduction

Neural network analysis is an algorithmic process that relies on mathematical models used to describe the functionality of the human brain. The human brain consists of millions of neurons which are used to communicate information. A simplistic mathematical model for the functionality of these structures is constructed by interconnecting processing elements that are analogous to the "cellular units of the nervous system" [2, p. 1]. These elements are organized into successive layers that are interconnected through an extensive network of synaptic junctions. These junctions serve as transmission lines through which individual elements communicate. Each element receives information from several input paths and processes the information using "a continuous function of the combined input" [2, p. 1].

Most neural networks consist of three layers which vary in the number of processing elements they contain. The first layer of processing elements forms the input buffer and accepts numerical values that describe an event. The last layer is the output buffer and "holds the response of the network to a given input" [2, p. 3]. In both of these layers, the processing elements each apply an activation function to the weighted summation of

values transmitted across the input junctions. For reasons that will be discussed later in this chapter, a sigmoid function is usually chosen for this purpose.

The true learning power of neural networks, however, results from the ability to adjust the distribution of weights acting on the output values of the processing elements. These weights can be adjusted such that the network is able to "identify and categorize" input data. Fig. 3 shows a single processing element located in the middle layer.

$$Ij = \sum_I WjXi$$

$$Y_j = f(I_j)$$
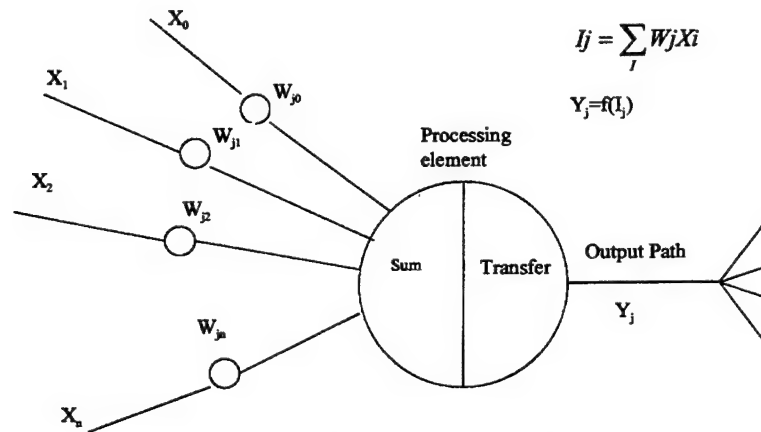
Fig. 3 Model for neural processing element [2, p. 3]

For a back-propagation neural network, the values assigned to the input layer are normalized within a network range. These normalized values are then weighted by an appropriate factor, summed at the hidden layer, and applied to a sigmoidal function. Undergoing a similar process at each succeeding layer, the values are propagated through the network until a response arrives at the output buffer. The associated weights between elements can then be adjusted by comparing this response to the desired output from a known training set. This procedure adjusts the weights such that each element becomes

trained to trigger only when certain characteristic patterns are present in the input buffer.

Hence, when the network receives an input not included in the training set, the output

will be matched to the training set vector which has the most similar pattern.

Now that the architecture and basic concepts of neural network processing have been

introduced, the mathematics of the training algorithm will be discussed.

(b) Network Training

The following notation is used to describe the back-propagation training algorithm [2,

pp. 64-67]:

$X_j^{[s]}$ value at the output of the jth neuron in layer s

$w_{ji}^{[s]}$ associated weight on the connection joining the ith neuron in layer (s-1) to the jth neuron in layer s

Additionally, a superscript "o" listed in brackets refers to the output layer of the network.

To begin, let the global error at the output of the network be represented by E such that:

$$E = 0.5\sum_k (d_k - o_k)^2 \tag{2.2.1}$$

where d_k is the desired k^{th} output and o_k is the actual k^{th} output. The local error at a

processing element k in the output layer can then be defined as the negative partial

derivative of the global error with respect to $I_k^{[o]}$ where $I_k^{[o]}$ represents the sum of the

weighted inputs to the kth neuron in the output layer prior to applying the activation

function f (Eq. 2.2.2). Using Eq. 2.2.1 and Eq. 2.2.2, the local error at element k

becomes the derivative of the activation function, f, multiplied by the difference between

the desired and actual output (Eq. 2.2.3).

$$X_j^s = f(\sum_i w_{ji}^{[s]} X_i^{[s-1]}) = f(I_j^{[s]}) \text{ or } X_k^o = f(I_k^o) \qquad (2.2.2)$$

$$e_k^o = -\frac{\partial E}{\partial I_k^o} = -\frac{\partial E}{\partial X_k} \cdot \frac{\partial X_k}{\partial I_k} = (d_k - o_k) \cdot f'(I_k) \qquad (2.2.3)$$

$$\text{where } X_k^o = o_k$$

The derivative of the activation function serves as a scaling factor and is multiplied by the summation of the errors being propagated back to the single k^{th} neuron. Eq. 2.2.4 (obtained from [2 p. 65]) reflects the error associated with the output of an element j located in layer s.

$$e_j^{[s]} = f'(I_k) \cdot \sum_k (e_k^{[s+1]} \cdot w_{kj}^{[s+1]}) \qquad (2.2.4)$$

The scaling factor imposed by the derivative of the activation function causes the error to converge to zero. This convergence is reinforced by choosing an activation function that maps the output of each element into a specified range. Because a sigmoidal function performs this mapping process over a zero to one range, it is usually used for the activation function (Eq. 2.2.5):

$$f(z) = (1.0 + e^{-z})^{-1} \qquad (2.2.5)$$

$$f'(z) = f(z)(1 - f(z)) \qquad (2.2.6)$$

Combining Eq. 2.2.4, 2.2.5 and 2.2.6, the local error can be written as:

$$e_j^{[s]} = X_j^{[s]}(1.0 - X_j^{[s]}) \cdot \sum_k e_k^{[s+1]} \cdot w_{kj}^{[s+1]} \qquad (2.2.7)$$

The critical parameter passed back is the local error $e_j^{[s]}$. This parameter is used to minimize the global error function, Eq. 2.2.1, which is used to judge the overall performance of the network. This minimization process is accomplished by adjusting the associated weights between processing elements via a gradient descent algorithm.

$$\Delta w_{ji}^{[s]} = -lcoef\left(\frac{\partial E}{\partial w_{ji}^{[s]}}\right) \tag{2.2.8}$$

The gradient descent method adjusts the weights by a small differential that corresponds in size and direction to the negative gradient of the error surface. The learning coefficient, *lcoef*, in Eq. 2.2.8 is a parameter for controlling this adjustment.

For the purpose of understanding the gradient descent algorithm and how it is used to locate a minimum of the global error function, a network consisting of only a single input and output processing element will be considered. A small network of this size is sufficient for describing the mathematics of the process. Such a network would contain only one connecting weight, and might produce a global error function similar to that seen in Fig. 4.
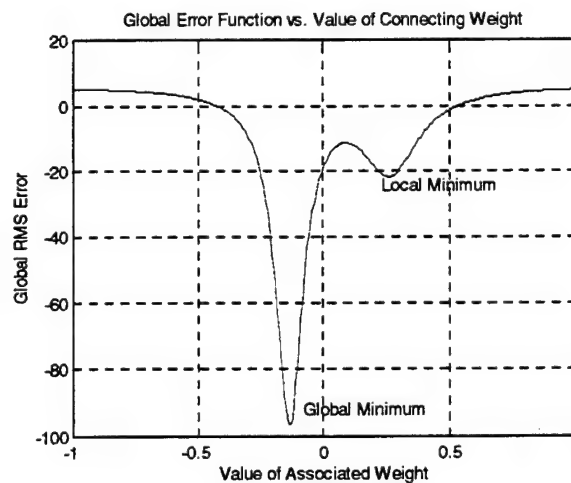

Fig. 4 Graph showing an arbitrary global error function

The gradient descent method uses the derivative of the global error function $\left(\frac{\partial E}{\partial w_{ji}}\right)$ to produce a differential increment that adds to the original weight. This process is iterative and produces a new weight that converges toward a minimum of the global error function according to:

$$\frac{\partial E}{\partial w_{ji}^{[s]}} = (\frac{\partial E}{\partial I_j^{[s]}}) \cdot (\frac{\partial I_j^{[s]}}{\partial w_{ji}^{[s]}}) = -e_j^{[s]} \cdot X_i^{[s-1]} \tag{2.2.9}$$

$$I_j^{[s]} = \sum_i (w_{ji}^{[s]} \cdot X_i^{[s-1]}) \tag{2.2.10}$$

$$\frac{\partial I_j^{[s]}}{\partial w_{ji}^{[s]}} = X_i^{[s-1]} \text{ where } i \text{ is a constant.} \tag{2.2.11}$$

$$\Delta w_{ji}^{[s]} = lcoef (e_j^{[s]} \cdot X_i^{[s-1]}) \tag{2.2.12}$$

The learning coefficient introduced in Eq. 2.2.12 is an additional parameter for changing the magnitude of the delta weight, Δw_{ji}. If the learning coefficient is set too high, the weights may oscillate about a minimum and never converge. If the learning coefficient is set too small, the network may converge to the first local minimum the gradient descent method encounters, and a more optimal solution may have been overlooked. Because there is no algorithm for defining an optimum learning coefficient, the value must be determined through a trial and error process. It is also important to note that while a local minimum does not represent a fully optimized weight distribution, this minimum may still provide results that are acceptable within a preset error criterion.

An additional term may also be added to the delta weight to reinforce the general trend of the step size and prevent oscillatory behavior. This term is referred to as the momentum.

$$\Delta w_{ji}^{[s]} = lcoef (e_j^{[s]} \cdot X_i^{[s-1]}) + momemtum \cdot \Delta w_{ji}^{[s]} \tag{2.2.13}$$

"Changing the weights as a linear function of the partial derivative makes the assumption that the error surface is locally linear" [2, p. 68]. This assumption does not hold at points of high curvature but can be corrected by adding a momentum term.

While the learning coefficient and momentum term force the global error function to converge to a minimum, the locality of this minimum cannot be determined. No algorithm exists for insuring that the weights have been adjusted to the values associated with the global minimum. Because of this, the weights must be adjusted through an iterative process until the network is able to correctly classify all sets of data contained in the training set.

3. Overview

In Section 1, the speech coding techniques were developed for finding the formant frequencies and cepstrum. These coding processes can now be combined with the neural network training algorithm described in Section 2 to phonetically classify a given speech signal. Before the network can be trained and the weights adjusted, a set of training data must first be developed. Chapter 3 describes the speech coding analysis for producing this data using the phoneme "e" as in "bet."

Chapter III: Phoneme Recognition and Simulation Results

The speech coding process outlined in Chapter II was applied to each of 40 different phonemes. Depending on the acoustical characteristics of the phoneme, the results of the coding process varied. For example, voiced speech produces a strong spike at the pitch frequency. For this fact, the voiced phoneme "e" was chosen to demonstrate the results of the coding process. Results produced by other phonemes are similar, except the formant frequencies are shifted and the cepstral spike varies in height.

1. MATLAB and NeuralWorks Simulation

An analog speech signal is first sampled using a 16-bit analog-to-digital converter and stored as a binary 'wav' file. For the voiced phoneme "e", the sampled signal appears as a periodic function and is represented in Fig. 5. This sampled signal consists of several hundred discrete points that are passed as a data vector into MATLAB.

Within MATLAB, the speech signal is segmented into 30 msec time slices/records and multiplied by a Hamming window. The results of this process are shown in Fig. 6. A record length of 30 msec was chosen because the vocal tract can be considered to be unchanging for that small duration of time, yet the periodicity of the waveform is still captured. Additionally, adjacent formant frequencies are often sufficiently close that a Hamming window is needed to prevent less prominent frequency characteristics from being masked by side band leakage. For this reason, a Hamming window was chosen. After applying this data window, the gated speech signal is padded with zeros to a power
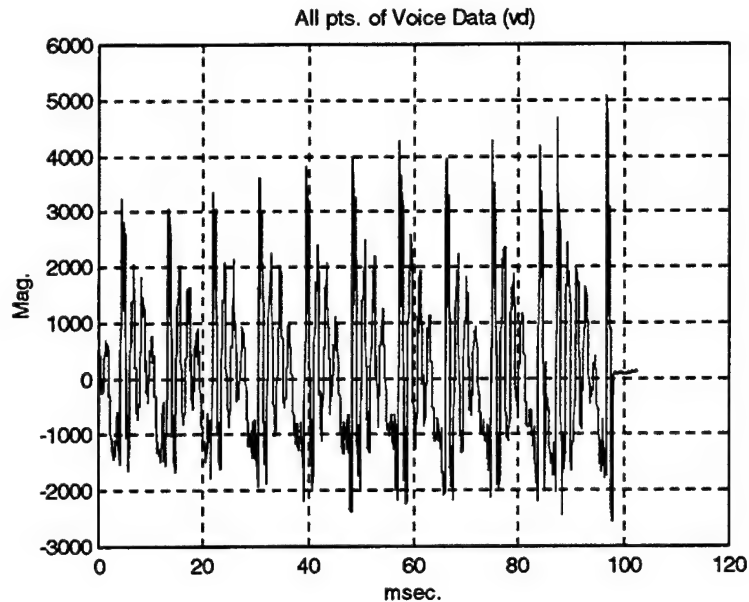
Fig. 5 Voice data for the phoneme "e" as in "bet"

of 2, normalized to unity RMS, and transformed using the radix-2 FFT (all MATLAB

programs are included in the Appendix). The result of this process is shown in Fig. 7

and represents the initial step of cepstral analysis.

In continuing this process, a second Fourier analysis is then applied to the logarithm of

the frequency spectrum. The resultant function is called the cepstrum and is shown in

Fig. 8. An important characteristic that results from this process is the occurrence of a

spike at each harmonic of the pitch frequency [4, p. 362]. For the voiced phoneme "e",

this initial spike occurs at 8 msec and corresponds to a pitch of 125 Hz.

To complete the speech coding process, a rectangular data window is applied to the

real part of the cepstrum to separate the vocal tract response from the excitation source,

thus revealing the formant frequencies (Fig. 9). These frequencies which appear as

humps in Fig. 9 shift depending on the position of the vocal tract.

**Fig. 6 Time slice of the phoneme "e"
with Hamming window applied**

**Fig. 7 Amplitude spectrum for 30 msec
time slice of phoneme "e"**

**Fig. 8 Cepstrum for 30 msec
time slice of the phoneme "e"**

**Fig. 9 Formant estimation for 30 msec
time slice of the phoneme "e"**
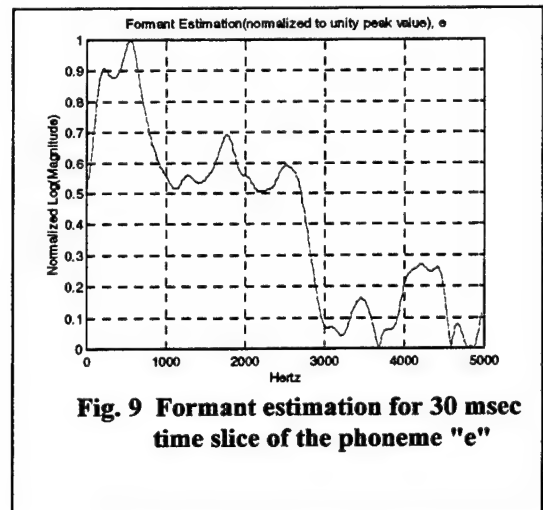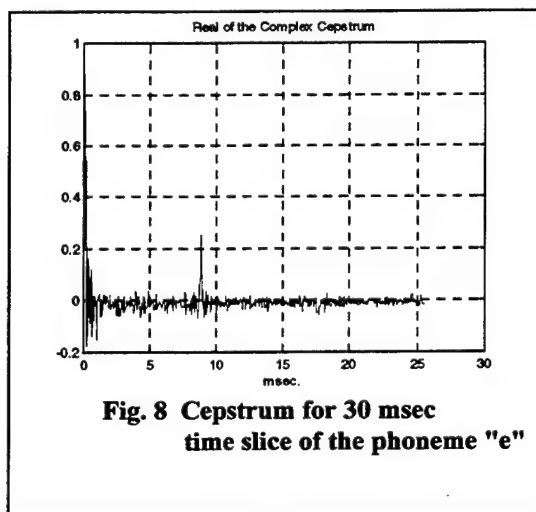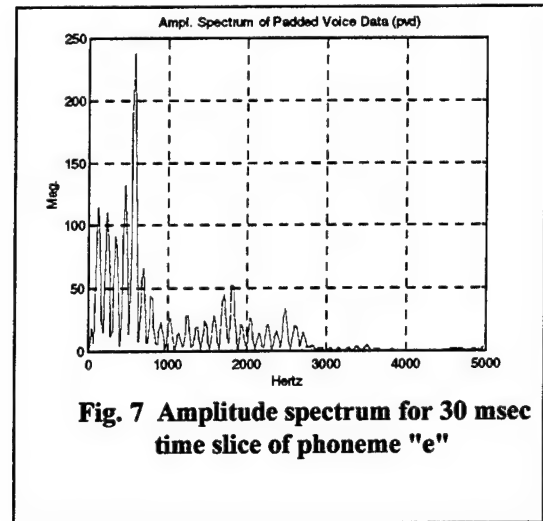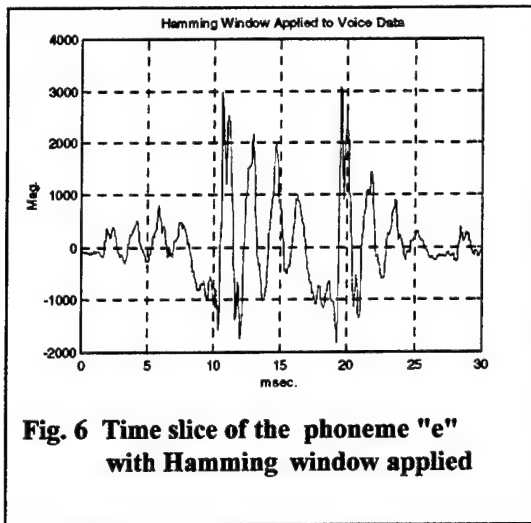
Finally, the cepstrum and formant estimation are concatenated into a single data vector

that is stored as an ASCII file and is represented in Fig. 10. This vector serves as input

to the back-propagation neural network.

Network INPUT for e.wav

Fig. 10 Network Input
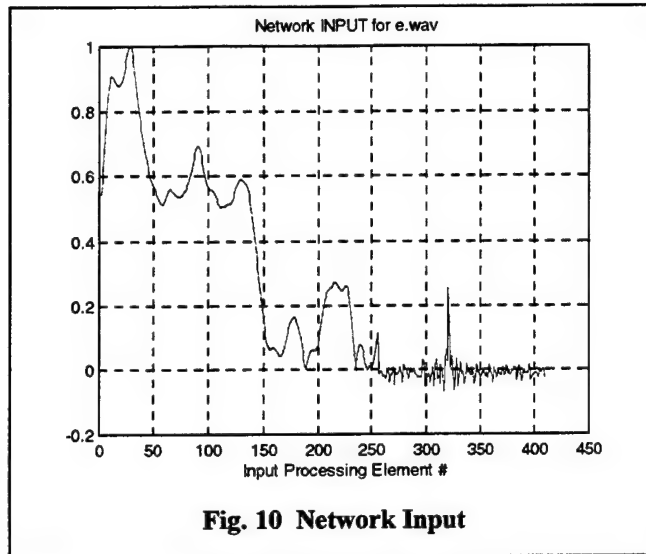
In order to provide training data to the network, six data banks were created to have

multiple examples of each phoneme. These data banks were prepared by segmenting

phonemes from various words and writing the data to a 'wav' file. In each case, the

particular phoneme being segmented was over stressed to insure that the resonance

characteristic of the vocal tract was emphasized. This speech coding process was

followed for each phonetic pronunciation in the data banks. In addition, the data vector

containing the cepstrum and formant estimation was appended to contain an additional

vector that described the desired output of the network. This additional vector allowed

the network to compare its actual output to the desired results and correspondingly adjust

the associated weights between elements. This process can be observed in NeuralWorks

by examining the convergence of the global error function, Eq. 2.2.1. This function is

proportional to the square of the Euclidean distance between the desired output and the

actual output of the network for a particular input pattern [2, p. 69]. The error function is

displayed as a strip chart that is labeled "RMS Error" and can be seen in Fig. 11. In

addition, a graphical matrix that quantifies the classification rate of the network is also displayed. This rate describes the ability of the network to correctly classify the training data. By observing these parameters and their convergence, the performance of the network can be partially judged. The most conclusive test, however, is attained by testing the network on speech data that is not included in the training set. While the network was trained on three data banks containing multiple examples of each phoneme, a similar number of data banks was reserved for testing.
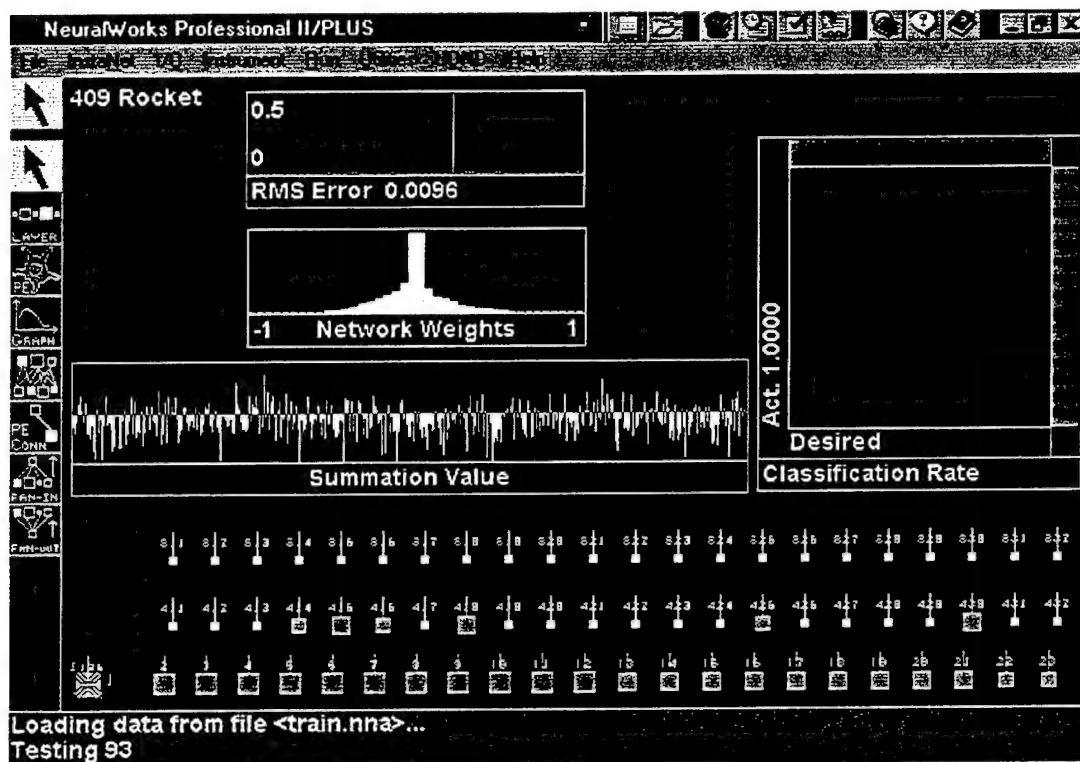
Fig. 11 NeuralWorks display of a back-propagation neural network trained to phonetically classify time slices of a speech signal. The network consist of 409 inputs that represent the frequency response of the vocal tract and the cepstrum. The hidden layer contains 400 processing elements and is fully connected to an output layer of 40 elements. These 40 elements each correspond to one phoneme.

2. Network Construction and Local Optimization

When attempting to optimize any neural network, several design variables become imperative to the performance of the system. The most important of these network variables include:

- Input Data
- Number of processing elements
- Number of hidden layers
- Connections between processing elements/layers
- Learning coefficient
- Network range/Gain
- Momentum term

Because there is no algorithm for determining a globally optimized value for each of these variables, estimates must be assigned and then adjusted according to the RMS error and classification rate. The choice of these estimates was based on specific considerations that will now be discussed for each variable.

(a) Input Data

The saying "garbage in...garbage out" applies as much to neural networks as it does to all other facets of data processing [6, p. 64]. For this reason, the most crucial element of neural network computing involves choosing the input data such that the network is provided enough information to effectively map the input to the output. The input for the network constructed in this project contains the formant estimations of the vocal tract coupled with a segment of the time domain cepstral function. Because the cepstrum produces a spike at the pitch frequency for voiced speech, the network is able to further define the resonant characteristics that distinguish different phonetic pronunciations. Although this network input does produce defining results for different pronunciations, there are, however, still several variables entwined in the speech coding process that may

create some ambiguity between phonemes. Some of these variables include the sampling rate, record length, windowing size applied to the cepstrum, and the number of ensemble averages.

◊ A sampling rate of 22.05 kHz was used. Accounting for the bandlimited characteristic of speech, this choice exceeded the Nyquist sampling rate which specifies that a signal must be sampled at a frequency of at least twice the bandwidth. Sampling at the minimum Nyquist rate would have given insufficient resolution in the frequency domain.

◊ Considering again that speech is bandlimited to approximately 5 kHz, the formant estimations were extracted for this frequency range. It was assumed that any formant frequencies that occurred outside this bandwidth resulted from transient sounds that provided no defined acoustical characteristics, and thus were ignored.

◊ Background noise is present in the sampled signal and introduces a degree of ambiguity in the formant estimation and cepstrum. This ambiguity may appear as a slight shift in the formant frequencies or a small change in amplitude of the cepstral spike. By averaging several records or time slices, this ambiguity becomes negligible. This process is known as ensemble averaging and improves results by averaging several records over a given expanse of time. While it may seem that the classification rate would naturally improve by continually increasing the number of ensemble averages, a phonetic pronunciation produced over a short period of time may actually be overlooked. Hence, the number of ensemble averages was confined to six.

◊ Each record length was chosen to represent 30 msec time slices of the speech signal. The window size was kept as short as possible so as to minimize the variations of speech parameters across the analysis interval. "The longer the window, the greater

the variation from beginning to end" [4, p. 377]. Additionally, each successive slice overlapped the preceding slice by 15 msec. This overlap was introduced to allow the number of ensemble averages to be increased.

◊ The windowing size applied to the cepstrum was chosen as 200 data points for a sampling rate of 22.05 kHz. This windowing size was chosen by making comparisons with various formant estimations produced by different N-point moving average filters, as discussed in Chapter II, Section 1.

(b) Number of Processing Elements

Using data banks that contained examples of 40 phonemes spoken by an adult male, a complete set of training data was prepared. From this data set, the number of elements in the input layer was determined. The input layer varied in length depending on the record size; but for a sampling rate of 22.05 kHz and a time slice of 30 msec, an input buffer of 409 elements was necessary to capture the formant frequencies and first cepstral spike. Of these 409 elements, 255 accounted for the formant frequencies under 5 kHz while the remaining 154 were used as a buffer to accept the sampled time data surrounding the first cepstral spike which occurs at 125 Hz. Accounting for a possible 50 Hz deviation from this estimation, 154 processing elements were used. These elements account for an initial spike occurring between 5.7 and 13.3 msec in the cepstral domain.

The next consideration was deciding upon the architecture of the output layer. The structure of a back-propagation network is designed such that each element in the output layer corresponds to one category of data. Hence, the output layer was constructed using 40 processing elements such that each element was indexed to a specific phoneme. Table 3.2.1 shows all 40 phonemes and their corresponding assigned index. These index values became the identifier for determining which phoneme was represented in the input

buffer. To explain how this process works, consider a set of data being applied to the input buffer. This data is processed by the network and a corresponding response arrives at the output layer. The element in the output buffer containing the highest value specifies the category with which the input string is most closely associated.

The final architectural design problem involved choosing the number of processing elements in the hidden layer. Throughout testing, several networks were created using a hidden layer of varying length. Four-hundred processing elements in this layer were ultimately found to produce the best results according to minimum global error criterion. Note that this value may not represent an optimized value.

(c) Number of Hidden Layers

During initial testing, only one hidden layer was used. The other network variables were tested sequentially, and sufficient classification results were obtained without having to add further layers.

(d) Connections Between Processing Elements and Layers

In the back-propagation algorithm there are two techniques for interconnecting successive processing elements and layers. In the traditional back-propagation algorithm, every element is connecting to all elements in the next succeeding layer. NeuralWork's back-propagation algorithm, however, provides an option of connecting each element to all previous elements. Both of these variations were explored, and the traditional back-propagation network showed a lower RMS error and a better classification rate for newly

Index	Letter Representation	Example	Index	Letter Representation	Example
1	IY	beet	21	M	maybe
2	I	bit	22	N	no
3	E	bet	23	NG	rung
4	AE	bat	24	B	bat
5	A	hot,almost	25	D	dumb
6	ER	bird	26	G	gold
7	UH	but	27	P	peck
8	OW	tome	28	T	tone
9	OO	boot	29	K	cold
10	U	foot	30	V	vast
11	AI	buy	31	TH	think
12	OI	boy	32	Z	zip
13	AU	how	33	ZH	leisure
14	EI	bay	34	F	fun
15	AX	about	35	S	sun
16	DH	than	36	SH	shop
17	W	win	37	H	hat
18	L	lamp	38	DZH	jam
19	R	rage	39	TSH	chip
20	Y	yes	40	null	-----

Table 3.2.1 English phonemes and assigned index

presented test data. From these results, it was found that the traditional back-propagation network was able to successfully classify phonemes.

(e) Learning Coefficient

The learning coefficient is a parameter that is used to control the size of the delta weight in the learning/training process (Eq. 2.2.8). Additionally, the learning coefficient works in conjunction with two other variables, the transition point and the learning coefficient ratio. Changing any of these three values effects the learning process and produces similar results. Table 3.2.2 lists the values used for each of these variables. These values were corroborated with other test results that were obtained from previous research into neural networks and vowel recognition [7, p. 38].

LAYER	LCoef
INPUT Layer	0.300
Hidden Layer	0.250
OUTPUT	0.150

Learning Coefficient Ratio = 0.5
Transition Point = 10,000

Table 3.2.2   LCoef, LCoef Ratio, and Transition Point

"The learning coefficient ratio sets the amount to divide the learning coefficient value by for the first transition. This ratio decays exponentially for subsequent transition points that are heuristically set to 3, 7, 15, and 31 times the initial transition point" [8, p. 25,26]. For example, if the learning coefficient is 0.3 and the learning coefficient ratio is 0.5, the network will train with a learning coefficient ratio of 0.3 for the first 10,000 training

iterations. After this training period, the value will change to 0.15 for the next 20,000 iterations.

During experimentation the learning coefficients and the learning coefficient ratio were fixed. The transition point was then changed and the RMS error for the network was observed for 100,000 training iterations. A transition point value of 10,000 was found to provide the best results.

(f) Network Ranges/Gain

Both the network range and the gain help to prevent an undesirable network condition known as saturation. When this occurs, the network effectively stops learning. To prevent this from happening, the data vector applied to the input buffer of the network is normalized and the sigmoidal activation function is slightly altered. To understand this process, first consider the gain. This value serves as a multiplier that is applied to the summation value of the processing element prior to computation of the activation function, f.

$$f(z) = (1.0 + e^{-GAIN \cdot z})^{-1} \qquad (3.2.1)$$

When the summation values become large, the transfer function produces an activation value of zero or one. "Consequently, the derivative becomes zero and the scaled error is always zero" [2, p. 73]. When this occurs, the processing elements stop learning.

$$f'(z) = f(z)(1 - f(z)) = 0 \text{ when } f(z) \text{ becomes saturated.}$$

Therefore:

$$X_j^s = f(\sum_i w_{ji}^{[s]} X_i^{[s-1]}) = f(I_j^{[s]}) = 0 \qquad (3.2.2)$$

$$e_j^{[s]} = X_j^{[s]}(1.0 - X_j^{[s]}) \cdot \sum_k e_k^{[s+1]} \cdot w_{kj}^{[s+1]} = 0 \qquad (3.2.3)$$

36

The network range and gain both serve to prevent this condition from happening.

The network range maps the input data within a specified set of values and helps to minimize the summation term at each element. This range was chosen such that the formant frequency estimations were normalized between zero and one. Similarly, the gain was adjusted to guard against saturation over a learning session of 100,000 iterations. NeuralWorks supports a graphical display that monitors the output summation of each processing element in the network (Fig. 11). Through 100,000 iterations, these summation terms were found to range in value between -8 and 8. To insure a monotonic mapping of the sigmoid function for this range of summation values, a gain value of 0.5 was chosen.

(g) Momentum

The momentum term is generally a percentage and was chosen to be a constant value of 40%. An increase in this value reinforces the general trend of descent in the linear approximation to the global error function and produces faster convergence toward a minimum. A smaller value, however, minimizes the error term in the vicinity of a minimum occurring at a point of high curvature. There are distinct advantages to both a small and large numerical momentum term. A value of 0.4 was chosen as a compromise to the benefits of both.

3. Best Network

After constructing several networks and experimenting with both the architectural design and the associated parameters governing the learning process, the classification results shown in Fig. 12 were obtained. These results were collected using test data that was drawn from a separate bank that had not been introduced to the network during the

Phoneme	Classification Rate (%)
IY	100
I	100
E	100
AE	100
A	100
ER	100
UH	100
OW	100
OO	100
U	100
AI	83.3
OI	100
AU	71.4
EI	100
AX	100
DH	100
W	100
L	100
R	100
Y	100

Phoneme	Classification Rate (%)
M	100
N	100
NG	93.8
B	0
D	0
G	0
P	0
T	0
K	0
V	100
TH	0
Z	100
ZH	100
F	0
S	100
SH	100
H	0
DZH	100
TSH	100
null	100

Fig. 12 Phoneme classification rate for newly introduced test data

training process. The data bank contained examples of all 40 phonemes; however, the actual number of test records per phoneme varied. This variation resulted from differences in the time length of each recorded phoneme. On average, 37 examples were presented to the network for each phoneme. As evident from Fig. 12, twenty-eight phonemes showed a hundred percent classification rate. For those phonemes, the network outperformed any voice processing system currently being marketed. Nine phonemes of the forty, however, showed a zero percent classification rate. This fact can be attributed to the acoustic characteristics intrinsic to each of the unidentified phonemes. With the exception of H, each of the unidentified phonemes (B, D, G, P, T, K, TH, and F) are transient, noncontinuant sounds which are produced by building up pressure behind a total constriction somewhere in the oral tract [4, p. 52]. As opposed to the other twenty-eight phonemes, these eight show no defined resonance within the oral cavity. Resonance does occur but is too variant in nature to define any characteristic formants. Because the neural network relied on the resonance characteristics of speech to identify

the various phonemes, the network was unable to correlate the formant estimation and cepstrum with the pronunciation of these eight phonemes.

The other phoneme showing a zero percent classification rate was H. This phoneme shows the same resonant characteristics of the vowel which follows its pronunciation. For this reason, H was always misidentified as a vowel.

With the exception of these nine phonemes, the NeuralWorks and MATLAB routines correctly classified all test words. For example, the network made the classification shown in Fig. 13 when presented the word "me."

```
» display(3)           % Function call to read the output of the neural network

The phonemes/indexes identified were:

0.5 sec:  21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21  1  1  1  1  1

» condense(3)          % Function call to remove repetitive phonemes in the output

The phonemes/indexes identified were:  21  1
```

Fig. 13 Example output from a MATLAB file that retrieves information contained in the output buffer of the network

The two functions, "display" and "condense," simply read the response that arrives in the output buffer of the neural network. The index numbers that these functions identify correspond to the phonemes listed in Table 3.2.1. The twenty-first element corresponds to the phoneme "M" and the first element corresponds to the phoneme "IY." This is the correct phonetic representation for the word "me." In the first routine, the output of the network is displayed using an average taken over three time slices. For instance, the first 60 msec of the test word was divided into three records each with 15 msec of overlap. The output of the network for these three records was then added together, and the corresponding record element containing the highest number identified the phoneme. This method of averaging records helps to prevent ambiguities in cases where two or

more elements in the output layer might contain very close values. In the second routine, the string of indexes is condensed and all repetitive occurrences are removed.

4. Overview and Concluding Remarks

In experimenting with both the speech coding process and the network design, several back-propagation neural networks were constructed. Each of these networks varied in either the coding data that the input buffer received or in the processing choices governing the actual design of the network. Some of these choices included the number of processing elements, connections between elements/layers, transition point, network range, and gain. By observing general trends imposed by changing each of these network variables, an optimized network was created that showed a hundred percent classification rate for twenty-eight phonemes. Nine phonemes, however, were impervious to correct classification. Because the network created a fundamental classification scheme that relied entirely upon the resonance characteristics of each phonetic pronunciation, any phoneme not showing a consistent frequency representation would be difficult to classify. This occurred with eight of the phonemes, these being B, D, G, P, T, K, TH, and F. One other phoneme, H, showed a formant estimation that possessed the same frequency characteristics as the vowel sound following its pronunciation. For this reason, it was always misclassified as a vowel.

Finally, it is important to remember that this paper presents one method for speech identification. Combining the speech coding and neural processing techniques introduced in this paper with other classification schemes might produce better results.

Chapter IV: Conclusion

1. Summary

A speaker-dependent back-propagation neural network was designed to recognize the phonemes present within a speech signal. This recognition system relies on information derived through a speech coding process that provides acoustical characteristics describing the physical nature of the vocal tract. This acoustical data consist of the formant frequencies concatenated with the first cepstral spike. Using this information, the network was trained to identify characteristic patterns that describe the phonemes present within a voice signal.

Because the recognition system is speaker-dependent, a list of phonemes was developed that could be interchanged to produce most spoken words in the English language. For these different phonetic pronunciations, a vocal tract model was created using a speech coding process that employed homomorphic deconvolution and cepstral analysis. This coding process was refined to provide the most clear description of the formant and pitch frequencies. Consequently, various experiments were performed to determine the record length and the size and type of data windows used during the process.

With a vocal tract model constructed for each phoneme, the network was trained on three data banks. Throughout this training process, the number of processing elements, learning coefficient, and gain were adjusted to produce a successful classification rate

and minimal global error. For newly introduced test data, the back-propagation neural network proved successful in identifying 28 phonemes with a classification rate of 100 percent. Additionally, three other phonemes (AI, AU, NG) were identified with an average classification rate of 83 percent. Each of these phonemes possessed strong resonant characteristics that could easily be deciphered from the formant estimation and first cepstral spike. The phonemes B, D, G, P, T, K, TH, F, and H, however, were misidentified. Because the vocal tract model relied entirely upon the frequency characteristics of the vocal tract, the model did not provide enough information to successfully map the input for these nine phonemes to the correct corresponding phonetic ouput. In particular, the phoneme H is not dynamic in nature but has a vocal tract shape that matches the vowel sound following its pronunciation. For this reason, H was always misidentified as the vowel following it.

When used for word classification, this network provided excellent results. Because the vocal tract transitions slowly in time, there is a naturally smooth transition between different phonetic pronunciations that allows ensemble averaging to be used. In addition to this method of averaging records, a statistical analysis was also applied to the output of the neural network to account for any ambiguities or redundancies that appeared. With the exception of any transient, noncontinuant sounds, the network correctly classified all the phonemes present within the test words.

2. Future Work

There are several areas where the classification scheme presented in this paper could be improved. One of these involves restructuring the input buffer to accept a new data vector containing the formant estimation, cepstrum, and filtering coefficients of the time domain speech signal. These filtering coefficients can be used with a random noise source to reproduce the original speech signal. These coefficients have been used in the past for identification and show distinct characteristics that help to further distinguish between phonemes.

3. Future Applications

The possible applications of a successful phonetic translator are almost unending. Using the index system introduced in Table 3.2.1, an analog speech signal could be transmitted as a 6-bit binary serial code. Although a synthetic voice decoder would be necessary to hear the reproduced signal, the information content would remain unchanged. Other benefits could include real-time interaction with machines, simple voice to text processing, or linguistic translators. All of these systems have the advantage of using a method of interface that is natural, inexpensive, and unspecialized.

References

[1] Andreas Spanias and Edward M. Painter, "A Software Tool for Introducing Speech Coding Fundamentals in a DSP Course," *IEEE Transactions on Education*, Vol. 39, No. 2, p. 143, May 1996.

[2] NeuralWare, Inc., *Neural Computing*, NeuralWare Publishing, 1995.

[3] Gordon Pelton, *Voice Processing*, McGraw-Hill, 1993.

[4] Lawrence Rabiner and Ronald Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, 1978.

[5] John Proakis and Dimitris Manolakis, *Digital Signal Processing 3rd Edition*, Prentice-Hall, 1996.

[6] Karla Yale, "Preparing the Right Data Diet for Training Neural Networks," *IEEE Spectrum*, Vol. 34, No. 3, pp. 64-66, Mar. 1997.

[7] Louiza Sellami, "Vowel Recognition in Adaptive Neural Networks, " Master Thesis, Electrical Engineering Dept., Univ. of Maryland, 1988.

[8] NeuralWare, Inc., *Reference Guide*, NeuralWare Publishing, 1995.

Appendix: MATLAB Programs

1. Setpath.m

```
% The following M-file adds the NeuralWare
% working directory(c:\nw2v523) to the
% path of the Matlab working directory.
%
P=path;
path(P,'c:\nw2v523');
P=path;
path(P,'c:\matlab\working2\wavfile');
path
clear P;
```

2. Setvars.m

```
% The following program sets the variables associated with
% the speech coding process that prepares the information
% being passed to input layer of the neural network for
% both training and testing.
%
% Set parameters
% d  = msec. per time slice
% ws = window size to be applied to cepstrum
% N  = # of output elements
% n  = # of input elements
% 'low' & 'high' pertain to the network ranges for input into the neural
network
% nv = number of ensemble averages; nv = 1 produces no averaging
clear;
load vars;
fprintf('\n\nCURRENT PARAMETER SETTINGS:\n\n');
fprintf('Sampling Rate = %2.2fkHz\n', vars(1));
fprintf('msec. per time slice(d) = %d\n', round(vars(2)/vars(1)));
fprintf('Overlaping time(msec) between consecutive slices = %d\n',
round(vars(3)/vars(1)));
fprintf('Window size applied to cepstrum(ws) = %d\n', vars(4));
fprintf('# of output elements in network(N) = %d\n', vars(5));
fprintf('# of input elements(n) = %d\n', vars(6));
fprintf('Network range minimum(low) = %1.1f\n', vars(7));
fprintf('Network range maximum(high) = %1.1f\n', vars(8));
fprintf('# of ensemble averages(nv) = %d\n\n', vars(9));
%
vars(1)=22.05;            % kHz
T=1/(fsamp);             % mSec
ncomp=0;            % Number of matched filter comparisons
% If ncomp is changed, the same alteration must be made in
% m-files train.m and trial.m
%
d = input('Enter value for d: ');
if isempty(d)==0
      vars(2) = fix(d*vars(1));
end
%
vars(3)=fix(vars(2)/2);
%
ws = input('Enter value for ws: ');
if isempty(ws)==0
      vars(4) = ws;
end
%
N = input('Enter value for N: ');
if isempty(N)==0
      vars(5) = N;
end
%
low = input('Enter value for low: ');
if isempty(low)==0
      vars(7) = low;
end
```

```
%
high = input('Enter value for high: ');
if isempty(high)==0
      vars(8) = high;
end
%
nv = input('Enter value for nv: ');
if isempty(nv)==0
      vars(9) = nv;
end
%
pitch=input('Enter pitch: ');
if isempty(pitch)==0
      vars(10)  = pitch;
      vars(11) = fix((1/(pitch+50)*(1000*fsamp)));
      vars(12) = fix((1/(pitch-50)*(1000*fsamp)));
end
%
% Compute n:
exp_2=ceil(1/log10(2) * log10(vars(2)));
nz = 2^(exp_2);
deltaF=1000/(nz*T);
vars(6)=fix(5000/deltaF)+vars(12)-vars(11)+1+ncomp;
%
%
fprintf('\n\nNEW PARAMETER SETTINGS:\n\n');
fprintf('Sampling Rate = %2.2fkHz\n', vars(1));
fprintf('msec. per time slice(d) = %d\n', round(vars(2)/vars(1)));
fprintf('Overlaping time(msec) between consecutive slices = %d\n',
round(vars(3)/vars(1)));
fprintf('Window size applied to cepstrum(ws) = %d\n', vars(4));
fprintf('# of output elements in network(N) = %d\n', vars(5));
fprintf('# of input elements(n) = %d\n', vars(6));
fprintf('Network range minimum(low) = %1.1f\n', vars(7));
fprintf('Network range maximum(high) = %1.1f\n', vars(8));
fprintf('# of ensemble averages(nv) = %d\n', vars(9));
fprintf('Speakers Pitch Frequency = %dHz\n\n', vars(10));
%
save vars;
```

3. Voice_1.m

```
% voice_1.m
% Read voice data created by Sound Blaster's 16 ADC
% Board. The sampling rate is 22.05KHz and the
% data settings are: Mono, 16 bit signed integer
% words/samples.
% This program time gates the data and plots the
% following:
% (1) Voice Data Array
% (2) Time Gated Data with Windowing Applied
% (3) Gated Data with Padding
% (4) Normalized Data
% (5) Amplitude Spectrum
% (6) Energy Spectral Density
% (7) Real of the Complex Cepstrum
% (8) Formant Estimation
% (9) Network INPUT
% Written by MIDN 1/c St. George
%
clear;
%
% Set parameters
% ws =       window size to be applied to cepstrum
% 'low' & 'high' pertain to the network ranges for
%       input into the neural network
% N-point moving average filter
%
load vars;
fsamp=vars(1);     % sampling rate in kHz
T=1/fsamp;   % T in msec.
ws=vars(4);
low=vars(7);
high=vars(8);
N=6;
tau1=vars(11);
tau2=vars(12);
%
% Read the Raw Return data.
%
test_fn=input('Input the File Name(without extension)=','s');
test_fn_ext=[test_fn,'.wav'];
vd=wavread(test_fn_ext);
%
%
% String arrays for the plots
%
c=fix(clock);
tp1=sprintf(', USNA %d/%d/%d %d:%d',c(1),c(2),c(3),c(4),c(5));
%
%
% Now Show vd array
%
x_t=0:T:(length(vd)-1)*T;
% figure(1);
subplot(111);
```

```
plot(x_t',vd,'b');
grid;
xlabel('msec.');
ylabel('Mag.');
t=['All pts. of Voice Data (vd)'];  %, ',test_fn,tp1];
title(t);
figure(gcf);
ppn=[test_fn,'_vd'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
%
%
% Now Time Gate this Voice Signal
%
%
st=input('Input the Starting Time(msec), st=');
ns=fix(st/T);
figure(gcf);
d=input('Input the Duration of the Time Slice(in msec.), d=');
np=fix(d/T);
ne=ns+np-1;
vd_gate(1:np)=vd(ns:ne);
%
% Window data using Hamming window
%
window=hamming(np);
window=window';
vd_w=window.*vd_gate;
%
% figure(2);
plot(x_t(1:np),vd_w,'b');
grid;
xlabel('msec.');
ylabel('Mag.');
t=[('Hamming Window Applied to Voice Data')];
title(t);
figure(gcf);
ppn=[test_fn,'_gw'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
% Find minimum padding to perform radix-2 fft
%
exp_2=ceil(1/log10(2) * log10(np));
nz = 2^(exp_2);
nzh=nz/2;
%
% Add padding
%
pvd=zeros(1,nz);
pvd(1:np)=vd_w(1:np);
%
% Now Show pvd array
%
% figure(3);
x_t=0:T:(nz-1)*T;
```

```
plot(x_t(1:nz),pvd,'b');
grid;
xlabel('msec.');
ylabel('Mag.');
t=['Padded Voice Data (pvd)'];        %, ',test_fn,tp1];
title(t);
figure(gcf);
ppn=[test_fn,'_pd'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
%
% Now remove the average value and normalize the data to have
% unity rms value.
%
avg=mean(pvd(1:np));
pvd(1:np)=pvd(1:np)-avg;
temp=pvd(1:np).^2;
rms=mean(temp);
rms=rms^0.5;
pvd(1:np)=(1/rms)*pvd(1:np);
clear temp;
%
%
%
% Now show the normalized pvd array
%
% figure(4);
plot(x_t(1:nz),pvd,'b');
grid;
xlabel('msec.');
ylabel('Mag.');
t=['Normalized Voice Data (pvd)'];  %, ',test_fn,tp1];
title(t);
figure(gcf);
ppn=[test_fn,'_nm'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
%
% Now compute the FFT of padded voice data.
%
F=1000*fsamp/nz;   % F in HZ
PVD=fft(pvd);
x_f=0:F:(nz-1)*F;
band=fix(5000/F);
%
% figure(5);
plot(x_f(1:band), abs(PVD(1:band)),'b');
grid;
xlabel('Hertz');
ylabel('Mag.');
t=['Ampl. Spectrum of Padded Voice Data (pvd)'];        %, ',test_fn,tp1];
title(t);
```

```
figure(gcf);
ppn=[test_fn,'_am'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
% Compute the Energy Spectral Density of PVD
%
%
% figure(6);
plot(x_f(1:nzh),20*log10(abs(PVD(1:nzh))),'b');
grid;
xlabel('Hertz');
ylabel(' Energy Spectral Density(Decibels) ');
t=['Energy Spectral Density'];
title(t);
figure(gcf);
ppn=[test_fn,'_sd'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
%
% Find the real of the complex cepstrum
%
h = PVD;
% logh = real(log(PVD + 0.01)); --> identical statement to line below
logh = log(abs(h)+0.01);
% an offset value of 0.01 was added to prevent the log of zero
cep = real(ifft(logh,nz));
%
% figure(7);
plot(x_t(1:nzh),cep(1:nzh),'b');
%
% With Fs=20.05 KHz, 200 sampling points will provide a spike
% for a pitch frequency above 100 Hz.
%
grid;
xlabel('msec.');
t=['Real of the Complex Cepstrum'];
title(t);
figure(gcf);
ppn=[test_fn,'_cp'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
%
% Show formant frequencies
% First Formant  {200-1200 Hz}
% Second Formant {500-3000 Hz}
%
%
cepstrum(nzh+1:nz)=cep(1:nzh);
cepstrum(1:nzh)=cep(nzh+1:nz);
window=hamming(ws);
```

```
window=window';
ht=cepstrum(nzh-ws/2+1:nzh+ws/2);
ht_w=window.*ht;
ht_p=zeros(1,nz);
ht_p(1:ws)=ht_w(1:ws);
HS=fft(ht_p);
HS_ABS=abs(HS);
%
% Normalize data to meet scaling and offset criteria.
%
scale=(high-low)/(max(HS_ABS)-min(HS_ABS));
offset=(max(HS_ABS)*low - min(HS_ABS)*high)/(max(HS_ABS) - min(HS_ABS));
X_adj=scale*HS_ABS+offset;
%
% Add first cepstral spike to normalized formant estimation
%
cep(tau1:tau2)=cep(tau1:tau2)+abs(min(cep(tau1:tau2)));
X_adj=[X_adj(1:band) cep(tau1:tau2)];
%
% N-Point moving average filter
%
% PVD_ABS=abs(PVD);
% for i = 1:band+N
%       PVD_ABS(i)=sum(PVD_ABS(i:i+N-1));
% end;
% PVD_ABS=log(PVD_ABS/N);
% scale=(high-low)/(max(PVD_ABS)-min(PVD_ABS));
% offset=(max(PVD_ABS)*low - min(PVD_ABS)*high)/(max(PVD_ABS) -
min(PVD_ABS));
% PVD_N=scale*PVD_ABS+offset;
%
%
%
% figure(8);
% subplot(211);
plot(x_f(1:band),abs(X_adj(1:band)),'-b');
grid;
xlabel('Hertz');
ylabel(' Normalized Log(Magnitude) ');
t=['Formant Estimation(normalized to unity peak value), ',test_fn ];
title(t);
% text(1000,0.95,'Cepstral Deconvolution','sc');
%
% subplot(212); plot(x_f(1:band),PVD_N(1:band),'-b');
% grid;
% xlabel('Hertz')
% ylabel(' Normalized Log(Magnitude) ');
% t=['Formant Estimation(normalized to unity peak value), ',test_fn ];
% title(t);
% text(1000,0.95,'Six-Point Moving Average Filter','sc');
figure(gcf);
ppn=[test_fn,'_fm'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
pause;
%
% Network Input
%
```

```
plot(X_adj, '-b');
grid;
xlabel('Input Processing Element #');
t=['Network INPUT for ',test_fn_ext ];
title(t);
figure(gcf);
ppn=[test_fn,'_ni'];
plot_fn=['c:\matlab\working2\plots_wp\',ppn];
eval(['print -dmeta ',plot_fn]);
```

4. Traindat.m

```
% The following M-file creates the train.nna
% file used by NeuralWare to train the back-
% propagation network.  All of the phonemes
% are equally represented in the training data.
% This is accomplished by using ensemble averaging
% over several time slices.
% Written by MIDN 1/c St. George
%
clear;
P=path;
dir_name=input('Enter directory containing voice data: ','s');
path(P,['c:\matlab\working2\', dir_name]);
if exist('train.nna')==2
      delete train.nna;
      fprintf('\nOverwriting file train.nna!\n\n');
else
      fprintf('\nCreating file test.nna!\n\n');
end;
train('iy', 1);
train('i',  2);
train('e',  3);
train('ae', 4);
train('a',  5);
train('er', 6);
train('uh', 7);
train('ow', 8);
train('oo', 9);
train('u', 10);
train('ai',11);
train('oi',12);
train('au',13);
train('ei',14);
train('ax',15);
train('dh',16);
train('w', 17);
train('l', 18);
train('r', 19);
train('y', 20);
train('m', 21);
train('n', 22);
train('ng',23);
train('b', 24);
train('d', 25);
train('g', 26);
train('p', 27);
train('t', 28);
train('k', 29);
train('v', 30);
train('th',31);
train('z', 32);
train('zh',33);
train('f', 34);
train('s', 35);
```

```
train('sh',36);
train('h', 37);
train('dzh',38);
train('tsh',39);
train('null',40);
path(P);
clear;
```

5. Train.m

```
function dv=train(fn, index)
% train.m
% Read voice data created by Sound Blaster's 16 ADC
% Board. The sampling rate is 22.05KHz and the
% data settings are: Mono, 16 bit signed integer
% words/sample.  This program time slices the data
% and writes the average formant frequencies to
% a training 'nna' file.
% The parameter 'index' represents the corresponding
% output element of the neural networkis which is
% driven high for the training data contained in fn.
% Written by MIDN 1/c St. George.
%
%
%
% Set parameters
% np =        # of data points composing time slice for
%      a sampling rate of 20.05 KHz
% no =        # of overlaping data points in consecutive
%      time slices
% ws =        window size to be applied to cepstrum
% N =         # of output elements
% n =         # of input elements
% 'low' & 'high' pertain to the network ranges for
%       input into the neural network
ncomp=0;
load vars;
np=vars(2);
no=vars(3);
ws=vars(4);
N=vars(5);
n=vars(6);
low=vars(7);
high=vars(8);
tau1=vars(11);
tau2=vars(12);
%
% Read the Raw Return data.
%
fn_ext=[fn, '.wav'];
vd=wavread(fn_ext);
%
% Now Time Gate this Voice Signal
%
% Find minimum padding to perform radiz-2 fft
%
exp_2=ceil(1/log10(2) * log10(np));
nz = 2^(exp_2);
nzh = nz/2;
band=n-(tau2-tau1+1);
fmt_avg=zeros(1,n);
num_records=floor((length(vd)-no)/no);
for ns=1:+no:no*num_records
```

```
ne=ns+np-1;
vd_gate(1:np)=vd(ns:ne);
%
% Window data using Hamming window
%
window=hamming(np);
window=window';
vd_w=window.*vd_gate;
%
% Add padding
%
pvd=zeros(1,nz);
pvd(1:np)=vd_w(1:np);
%
% Remove the average value and normalize the data
% to have unity rms value.
%
avg=mean(pvd(1:np));
pvd(1:np)=pvd(1:np)-avg;
temp=pvd(1:np).^2;
rms=mean(temp);
rms=rms^0.5;
pvd(1:np)=(1/rms)*pvd(1:np);
clear temp;
%
% Compute the FFT of padded voice data.
%
PVD=fft(pvd);
%
% Find the real of the complex cepstrum
%
h = PVD;
logh = log(abs(h)+0.01);
% an offset value of 0.01 was added to prevent
% the log of zero.
cep = real(ifft(logh,nz));
%
% Compute formant frequencies and
% frequency domain response of vocal tract model
% First Formant  {200-1200 Hz}
% Second Formant {500-3000 Hz}
%
%
cepstrum(nzh+1:nz)=cep(1:nzh);
cepstrum(1:nzh)=cep(nzh+1:nz);
window=hamming(ws);
window=window';
ht=cepstrum(nzh-ws/2+1:nzh+ws/2);
ht_w=window.*ht;
ht_p=zeros(1,nz);
ht_p(1:ws)=ht_w(1:ws);
HS=fft(ht_p);
HS_ABS=abs(HS(1:nzh));
%
% Normalize data to meet scaling and
% offset criteria.
%
scale=(high-low)/(max(HS_ABS)-min(HS_ABS));
```

```
        offset=(max(HS_ABS)*low - min(HS_ABS)*high)/(max(HS_ABS) -
min(HS_ABS));
        X_adj=scale*HS_ABS+offset;
        cep(tau1:tau2)=cep(tau1:tau2)+abs(min(cep(tau1:tau2)));
        X_adj=[X_adj(1:band)  cep(tau1:tau2)];
        if (index >= 1) & (index <= N)
               fmt_avg = fmt_avg+X_adj;
        end
end
if num_records == 0
        fprintf('\nNo training data was created for file ');
        fprintf(fn_ext); fprintf('\n\n');
elseif (index >= 1) & (index <= N)
        fmt_avg=fmt_avg/num_records;
        if ncomp > 0
               fmt_avg(n-ncomp+1:n) = check(vd');
        end;
        fmt_avg(n+1:n+N) = zeros(1,N);
        fmt_avg(n+index) = 1;
        fid = fopen('train.nna', 'a');
        fprintf(fid, '%1.8f ', fmt_avg(1:n+N-1));
        fprintf(fid, '%1.8f\n', fmt_avg(n+N));
        fclose('all');
end
```

6. Testdat.m

```
% The following M-file creates a test.nna
% file used by NeuralWare to test the back-
% propagation network.  All of the phonemes
% are represented.
% Written by MIDN 1/c St. George
%
clear;
P=path;
dir_name=input('Enter directory containing voice data: ','s');
path(P,['c:\matlab\working2\', dir_name]);
if exist('test.nna')==2
      delete test.nna;
      fprintf('\nOverwriting file test.nna!\n\n');
else
      fprintf('\nCreating file test.nna!\n\n');
end;
slices(1)=trial('iy');
slices(2)=trial('i');
slices(3)=trial('e');
slices(4)=trial('ae');
slices(5)=trial('a');
slices(6)=trial('er');
slices(7)=trial('uh');
slices(8)=trial('ow');
slices(9)=trial('oo');
slices(10)=trial('u');
slices(11)=trial('ai');
slices(12)=trial('oi');
slices(13)=trial('au');
slices(14)=trial('ei');
slices(15)=trial('ax');
slices(16)=trial('dh');
slices(17)=trial('w');
slices(18)=trial('l');
slices(19)=trial('r');
slices(20)=trial('y');
slices(21)=trial('m');
slices(22)=trial('n');
slices(23)=trial('ng');
slices(24)=trial('b');
slices(25)=trial('d');
slices(26)=trial('g');
slices(27)=trial('p');
slices(28)=trial('t');
slices(29)=trial('k');
slices(30)=trial('v');
slices(31)=trial('th');
slices(32)=trial('z');
slices(33)=trial('zh');
slices(34)=trial('f');
slices(35)=trial('s');
slices(36)=trial('sh');
slices(37)=trial('h');
```

```
slices(38)=trial('dzh');
slices(39)=trial('tsh');
slices(40)=trial('null');
fprintf('\n%d records written to test.dat\n\n', sum(slices));
save slices;
path(P);
clear;
```

7. Trial.m

```
function num_records=trial(fn)
% test.m
% Read voice data created by Sound Blaster's 16 ADC
% Board. The sampling rate is 22.05KHz and the
% data settings are: Mono, 16 bit signed integer
% words/sample.  This program time slices the data
% and writes the average formant frequencies to
% a testing 'nna' file.
% Written by MIDN 1/c St. George.
%
%
%
% Set parameters
% np =        # of data points composing time slice for
%      a sampling rate of 20.05 KHz
% no =        # of overlaping data points in consecutive
%      time slices
% ws =        window size to be applied to cepstrum
% N =        # of output elements
% n =        # of input elements
% 'low' & 'high' pertain to the network ranges for
%       input into the neural network
% nv = number of ensemble averages; nv = 1 produces
%       no averaging
ncomp=0;
load vars;
np=vars(2); % 400 data points approximately equals 20 msec.
no=vars(3);
ws=vars(4);
N=vars(5);
n=vars(6);
low=vars(7);
high=vars(8);
nv=vars(9);
tau1=vars(11);
tau2=vars(12);
%
% Read the Raw Return data.
%
fn_ext=[fn, '.wav'];
vd=wavread(fn_ext);
%
% Now Time Gate this Voice Signal
%
% Find minimum padding to perform radiz-2 fft
%
exp_2=ceil(1/log10(2) * log10(np));
nz = 2^(exp_2);
nzh = nz/2;
band=n-(tau2-tau1+1);
num_records=floor((length(vd)-no)/no);
for ns=1:+no:no*num_records
        ne=ns+np-1;
```

```
vd_gate(1:np)=vd(ns:ne);
%
% Window data using Hamming window
%
window=hamming(np);
window=window';
vd_w=window.*vd_gate;
%
% Add padding
%
pvd=zeros(1,nz);
pvd(1:np)=vd_w(1:np);
%
% Remove the average value and normalize the data
% to have unity rms value.
%
avg=mean(pvd(1:np));
pvd(1:np)=pvd(1:np)-avg;
temp=pvd(1:np).^2;
rms=mean(temp);
rms=rms^0.5;
pvd(1:np)=(1/rms)*pvd(1:np);
clear temp;
%
% Compute the FFT of padded voice data.
%
PVD=fft(pvd);
%
% Find the real of the complex cepstrum
%
h = PVD;
logh = log(abs(h)+0.01);
% an offset value of 0.01 was added to prevent
% the log of zero.
cep = real(ifft(logh,nz));
%
% Compute formant frequencies and
% frequency domain response of vocal tract model
% First Formant   {200-1200 Hz}
% Second Formant  {500-3000 Hz}
%
%
cepstrum(nzh+1:nz)=cep(1:nzh);
cepstrum(1:nzh)=cep(nzh+1:nz);
window=hamming(ws);
window=window';
ht=cepstrum(nzh-ws/2+1:nzh+ws/2);
ht_w=window.*ht;
ht_p=zeros(1,nz);
ht_p(1:ws)=ht_w(1:ws);
HS=fft(ht_p);
HS_ABS=abs(HS(1:nzh));
%
% Normalize data to meet scaling and
% offset criteria.
%
scale=(high-low)/(max(HS_ABS)-min(HS_ABS));
```

```
        offset=(max(HS_ABS)*low - min(HS_ABS)*high)/(max(HS_ABS) -
min(HS_ABS));
        X_adj=scale*HS_ABS+offset;
        cep(tau1:tau2)=cep(tau1:tau2)+abs(min(cep(tau1:tau2)));
        X_adj=[X_adj(1:band) cep(tau1:tau2)];
        %
        %
        ens(ceil(ns/no),1:n-ncomp)=X_adj(1:n-ncomp);
        %
        %
end;
if (num_records >= nv)
        % vd=[vd' zeros(1,(nv-1)*no)];
        for i=num_records+1:num_records+nv-1
                ens(i,1:n-ncomp)=zeros(1,n-ncomp);
        end;
        for i=1:num_records
                for j=1:nv-1
                        ens(i,1:n-ncomp)=ens(i,1:n-ncomp)+ens(i+j,1:n-ncomp);
                end
                if i <= num_records-nv+1
                        div=nv;
                else
                        div=num_records-i+1;
                end;
                ens(i,1:n-ncomp)=ens(i,1:n-ncomp)/div;
                % Correlation of matched function with all ensembles
                % start=(i-1)*no+1;
                % stop=start+(nv+1)*no-1;
                % Correlation of matched function with single record
                if ncomp > 0
                        start=(i-1)*no+1;
                        stop=start+np-1;
                        ens(i,n-ncomp+1:n)=check(vd(start:stop)');
                end;
                fid = fopen('test.nna', 'a');
                fprintf(fid, '%1.8f ', ens(i,1:n-1));
                fprintf(fid, '%1.8f\n', ens(i,n));
                fclose('all');
        end
elseif (num_records < nv)
        fprintf('\nToo few records exist in ');
        fprintf(fn_ext);
        fprintf(' to perform %d ensemble averages.\n', nv);
        if num_records == 0
                fprintf('No test data was created for file ');
                fprintf(fn_ext); fprintf('\n\n');
        else
                fprintf('Writing records to test file with %d ensemble
average(s)!\n\n', num_records);
                for i=num_records+1:num_records+nv-1
                        ens(i,1:n-ncomp)=zeros(1,n-ncomp);
                end;
                nv=num_records;
                % vd=[vd' zeros(1,(nv-1)*no)];
                for i=num_records+1:num_records+nv-1
                        ens(i,1:n-ncomp)=zeros(1,n-ncomp);
                end;
```

```
            for i=1:num_records
                  for j=1:nv-1
                        ens(i,1:n-ncomp)=ens(i,1:n-ncomp)+ens(i+j,1:n-
ncomp);
                  end;
                  if i <= num_records-nv+1
                        div=nv;
                  else
                        div=num_records-i+1;
                  end;
                  ens(i,1:n-ncomp)=ens(i,1:n-ncomp)/nv;
                  % Correlation of matched function with all ensembles
                  % start=(i-1)*no+1;
                  % stop=start+(nv+1)*no-1;
                  % Correlation of matched function with single record
                  if ncomp > 0
                        start=(i-1)*no+1;
                        stop=start+np-1;
                        ens(i,n-ncomp+1:n)=check(vd(start:stop)');
                  end;
                  fid = fopen('test.nna', 'a');
                  fprintf(fid, '%1.8f ', ens(i,1:n-1));
                  fprintf(fid, '%1.8f\n', ens(i,n));
                  fclose('all');
            end;
      end;
end
```

8. Class.m

```
function dv=class(N)
% N sets N-point filterning window (N >= 1)
% This M-File displays the clssification results
% for testing the network on a complete data bank
% of phonemes.
% Written by MIDN 1/c St. George
%
load test.nnr;
[num_rec, out]=size(test);
load slices.mat
if (nargin==1) & (num_rec==sum(slices))
      if (nargin==1) & N==1
            [Y,I]=max(test');
      elseif (nargin==1) & (N>1) & fix((N+1)/2)==(N+1)/2
            window=hamming(N);
            test_t=test';
            sum=zeros(1,out);
            pad=zeros(out,(N-1)/2);
            test_p=[pad test_t pad];
            for i=0:num_rec-1
                  for n=1:N
                        sum=sum+window(n)*test_p(out*i+(n-
1)*out+1:out*i+n*out);
                  end
                  sum_t=sum';
                  if i==0
                        test_f=sum_t;
                  else
                        test_f=[test_f sum_t];
                  end;
            end;
            [Y,I]=max(test_f);
      end;
      num_phone=length(slices);
      count=0;
      for i=1:num_phone
            corr_class=0;
            for j=1:slices(i)
                  if I(count+j)==i
                        corr_class=corr_class+1;
                  end;
            end;
            count=count+slices(i);
            if slices(i) ~= 0
                  id(i)=100*corr_class/slices(i);
            else
                  id(i)=slices(i);
            end;
      end;
      fprintf('\nClassification Results: \n\n');
      fprintf('IY   = %3.1f\n', id(1));
      fprintf('I    = %3.1f\n', id(2));
      fprintf('E    = %3.1f\n', id(3));
```

```
        fprintf('AE    = %3.1f\n', id(4));
        fprintf('A     = %3.1f\n', id(5));
        fprintf('ER    = %3.1f\n', id(6));
        fprintf('UH    = %3.1f\n', id(7));
        fprintf('OW    = %3.1f\n', id(8));
        fprintf('OO    = %3.1f\n', id(9));
        fprintf('U     = %3.1f\n', id(10));
        fprintf('AI    = %3.1f\n', id(11));
        fprintf('OI    = %3.1f\n', id(12));
        fprintf('AU    = %3.1f\n', id(13));
        fprintf('EI    = %3.1f\n', id(14));
        fprintf('AX    = %3.1f\n', id(15));
        fprintf('DH    = %3.1f\n', id(16));
        fprintf('W     = %3.1f\n', id(17));
        fprintf('L     = %3.1f\n', id(18));
        fprintf('R     = %3.1f\n', id(19));
        fprintf('Y     = %3.1f\n', id(20));
        fprintf('M     = %3.1f\n', id(21));
        fprintf('N     = %3.1f\n', id(22));
        fprintf('NG    = %3.1f\n', id(23));
        fprintf('B     = %3.1f\n', id(24));
        fprintf('D     = %3.1f\n', id(25));
        fprintf('G     = %3.1f\n', id(26));
        fprintf('P     = %3.1f\n', id(27));
        fprintf('T     = %3.1f\n', id(28));
        fprintf('K     = %3.1f\n', id(29));
        fprintf('V     = %3.1f\n', id(30));
        fprintf('TH    = %3.1f\n', id(31));
        fprintf('Z     = %3.1f\n', id(32));
        fprintf('ZH    = %3.1f\n', id(33));
        fprintf('F     = %3.1f\n', id(34));
        fprintf('S     = %3.1f\n', id(35));
        fprintf('SH    = %3.1f\n', id(36));
        fprintf('H     = %3.1f\n', id(37));
        fprintf('TSH   = %3.1f\n', id(39));
        fprintf('null = %3.1f\n', id(40));
        fprintf('\n\n');
elseif (nargin==1)
        fprintf('\nThe number of records contained in test.nnr\n');
        fprintf('does not match the number of time slices\n');
        fprintf('recorded by testdat.m\n\n');
end;
```

9. Voice_t.m

```
% voice_t.m
% Read voice data created by Sound Blaster's 16 bit ADC
% Board. The sampling rate is 22.05KHz and the
% data settings are: Mono, 16 bit signed integer words/sample.
% This program time slices the data and writes the average
% formant frequencies to either a training or testing 'nna'
% file.  An index of '0' signifies a testing file.
% An index >= 1  signifies a training data file in which the
% corresponding output element of the neural network(specified
% by the numerical value assigned the variable index) is
% driven high.
% Written by MIDN 1/c St. George.
%
%
clear;
%
% Set parameters
% np = # of data points composing time slice for a sampling rate of
20.05 KHz
% no = # of overlaping data points in consecutive time slices
% ws = window size to be applied to cepstrum
% N = # of output elements
% n = # of input elements
% 'low' & 'high' pertain to the network ranges for input into the neural
network
% nv = number of ensemble averages; nv = 1 produces no averaging
load vars;  % sampling rate in kHz.
fsamp=vars(1);
np=vars(2);
no=vars(3);
ws=vars(4);
N=vars(5);
n=vars(6);
low=vars(7);
high=vars(8);
nv=vars(9);
tau1=vars(11);
tau2=vars(12);
%
% Read the Raw Return data.
%
fn=input('Input the File Name(without extension): ','s');
index=input('Input phoneme index(Enter 0 to produce test data): ');
fn_ext=[fn, '.wav'];
vd=wavread(fn_ext);
%
%
% String arrays for the plots
%
c=fix(clock);
tp1=sprintf(', USNA %d/%d/%d %d:%d',c(1),c(2),c(3),c(4),c(5));
delx=int2str(fix(1000/fsamp));  % delx in micro sec.
%
```

```
%
% Now Show vd array
%
% Now Time Gate this Voice Signal
%
band=n-(tau2-tau1+1);
fmt_avg=zeros(1,n);
num_records=floor((length(vd)-no)/no);
for ns=1:+no:no*num_records
      %
      ne=ns+np-1;
      vd_gate(1:np)=vd(ns:ne);
      %
      % Window data using Hamming window
      %
      window=hamming(np);
      window=window';
      vd_w=window.*vd_gate;
      %
      %
      % Find minimum padding to perform radix-2 fft
      %
      exp_2=ceil(1/log10(2) * log10(np));
      nz = 2^(exp_2);
      nzh = nz/2;
      %
      % Add padding
      %
      pvd=zeros(1,nz);
      pvd(1:np)=vd_w(1:np);
      %
      %
      % Now remove the average value and normalize the data to have
      % unity rms value.
      %
      avg=mean(pvd(1:np));
      pvd(1:np)=pvd(1:np)-avg;
      temp=pvd(1:np).^2;
      rms=mean(temp);
      rms=rms^0.5;
      pvd(1:np)=(1/rms)*pvd(1:np);
      clear temp;
      %
      %
      % Now compute the FFT of padded voice data.
      %
      delf=int2str(fix(1e6*fsamp/nz));  % delx in mHz.
      PVD=fft(pvd);
      %
      %
      % Find the real of the complex cepstrum
      %
      h = PVD;
      logh = log(abs(h)+0.01);
      % an offset value of 0.01 was added to prevent the log of zero
      cep = real(ifft(logh,nz));
      %
      %
```

```
% Show formant frequencies and
% frequency domain response of vocal tract model
% First Formant  {200-1200 Hz}
% Second Formant {500-3000 Hz}
%
%
cepstrum(nzh+1:nz)=cep(1:nzh);
cepstrum(1:nzh)=cep(nzh+1:nz);
window=hamming(ws);
window=window';
ht=cepstrum(nzh-ws/2+1:nzh+ws/2);
ht_w=window.*ht;
ht_p=zeros(1,nz);
ht_p(1:ws)=ht_w(1:ws);
HS=fft(ht_p);
HS_ABS=abs(HS(1:nzh));
%
% Normalize data to meet scaling and offset criteria.
%
scale=(high-low)/(max(HS_ABS)-min(HS_ABS));
offset=(max(HS_ABS)*low - min(HS_ABS)*high)/(max(HS_ABS) -
min(HS_ABS));
X_adj=scale*HS_ABS+offset;
X_adj=[X_adj(1:band) cep(tau1:tau2)];
fmt_avg = fmt_avg+X_adj;
%
if index == 0
        ens(ceil(ns/no),1:n)=X_adj(1:n);
end
end
if (index == 0)
        if exist('test.nna')==2
                delete test.nna;
                fprintf('\nOverwriting file test.nna!\n');
        else
                fprintf('\nCreating file test.nna!\n');
        end;
elseif (index >= 1) & (index <= N)
        if exist('train.nna')==2
                fprintf('\nAppending file train.nna!\n');
        else
                fprintf('\nCreating file train.nna!\n');
        end;
end;
nr=num_records-(nv-1);
if (index == 0) & (nr >= 1)
        for i=1:nr
                for j=1:nv-1
                        ens(i,1:n)=ens(i,1:n)+ens(i+j,1:n);
                end
                ens(i,1:n)=ens(i,1:n)/nv;
                fid = fopen('test.nna', 'a');
                fprintf(fid, '%1.8f ', ens(i,1:n-1));
                fprintf(fid, '%1.8f\n', ens(i,n));
                fclose('all');
        end
elseif (index == 0) & (nr < 1)
        fprintf('\nToo few records exist in ');
```

```
        fprintf(fn_ext);
        fprintf(' to average %d ensembles!\n',nv);
        fprintf('Writing records to test file without ensemble
averaging\n\n');
        for i=1:num_records
                fid = fopen('test.nna', 'a');
                fprintf(fid, '%1.8f ', ens(i,1:n-1));
                fprintf(fid, '%1.8f\n', ens(i,n));
                fclose('all');
        end
end
if (index >= 1) & (index <= N)
        fmt_avg=fmt_avg/num_records;
        fmt_avg(n+1:n+N) = zeros(1,N);
        fmt_avg(n+index) = 1;
        fid = fopen('train.nna', 'a');
        fprintf(fid, '%1.8f ', fmt_avg(1:n+N-1));
        fprintf(fid, '%1.8f\n', fmt_avg(n+N));
        fprintf('\n%d record(s) copied to output file\n\n', 1);
        fclose('all');
elseif index == 0
        fprintf('\n%d record(s) copied to output file\n\n', nr);
end
```

10. Display.m

```
function dv=display(npt)
% N sets N-point filterning window (N >= 1)
% This M-File display the results taken from the
% output buffer/layer of the neural network.  A
% statistical analysis is applied to consecutive
% records to account for ambiguities that may
% arise.
% Written by MIDN 1/c St. George
%
load vars;
fsamp=vars(1);      % sampling rate in kHz.
T=1/fsamp;   % T in msec.
np=vars(2);
no=vars(3);
% the number of records comprising a 250 msec time interval
quart_sec=round(2*(250/T/np)-1);
%
load test.nnr;
[rec_num, out]=size(test);
if npt/2==fix(npt/2)
      N=npt-1;
else
      N=npt;
end;
if N==1
      [Y,I]=max(test');
elseif (N>1) & fix((N+1)/2)==(N+1)/2
      window=hamming(N);
      test_t=test';
      sum=zeros(1,out);
      pad=zeros(out,(N-1)/2);
      test_p=[pad test_t pad];
      for i=0:rec_num-1
            for n=1:N
                  sum=sum+window(n)*test_p(out*i+(n-
1)*out+1:out*i+n*out);
            end
            sum_t=sum';
            if i==0
                  test_f=sum_t;
            else
                  test_f=[test_f sum_t];
            end
      end
      [Y,I]=max(test_f);
end
%
fprintf('\nThe phonemes/indexes identified were: \n\n');
k=1;
while (k <= length(I))
      if length(k:length(I)) < quart_sec
            quart_sec = length(I)-k+1;
      end;
```

```
        fprintf('%1.2f sec:  ', 0.25*floor(k/round(2*(250/T/np)-1))+0.25);
        fprintf('%2.0f ', I(k:k+quart_sec-1));
        fprintf('\n');
        k=k+quart_sec;
end;
fprintf('\n');
```

11. Condense.m

```matlab
function dv=condense(npt)
% npt sets N-point filterning window (N >= 1)
%
% The output is CONDENSED and any repetitive
% or redundant phonemes are removed.
% Written by MIDN 1/c St. George
%
load vars;
fsamp=vars(1);      % sampling rate in kHz.
T=1/fsamp;  % T in msec.
np=vars(2);
no=vars(3);
% the number of records comprising a 250 msec time interval
quart_sec=round(2*(250/T/np)-1);
%
load test.nnr;
[rec_num, out]=size(test);
if npt/2==fix(npt/2)
      N=npt-1;
else
      N=npt;
end;
if N==1
      [Y,I]=max(test');
elseif (N>1) & fix((N+1)/2)==(N+1)/2
      window=hamming(N);
      test_t=test';
      sum=zeros(1,out);
      pad=zeros(out,(N-1)/2);
      test_p=[pad test_t pad];
      for i=0:rec_num-1
            for n=1:N
                  sum=sum+window(n)*test_p(out*i+(n-
1)*out+1:out*i+n*out);
            end
            sum_t=sum';
            if i==0
                  test_f=sum_t;
            else
                  test_f=[test_f sum_t];
            end
      end
      [Y,I]=max(test_f);
end;
%
cond_out=I(1);
d=1;
for i=2:rec_num;
      if cond_out(i-d)~=I(i)
            cond_out=[cond_out I(i)];
      else
            d=d+1;
      end
      end
```

```
end
clear I;
I=cond_out;
fprintf('\nThe phonemes/indexes identified were: \n\n');
k=1;
while (k <= length(I))
      if length(k:length(I)) < quart_sec
            quart_sec = length(I)-k+1;
      end;
      fprintf('%2.0f ', I(k:k+quart_sec-1));
      fprintf('\n');
      k=k+quart_sec;
end;
fprintf('\n');
```